Decentralized GPU Network with Proof of Rendering

Inferix Team (version August 2, 2024)

***Abstract*—This paper introduces Proof-of-Rendering (PoR) and its application in building Inferix's decentralized GPU network. Addressing DePIN Verification, the key challenge facing developers of decentralized physical infrastructure networks (DePIN) recently, Inferix has developed the Active Noise Generation (ANG), Active Noise Verification (ANV), and PoR algorithms. These algorithms are combined with a software layer that includes middleware and client SDK, facilitating connections between 3D creative data systems and the decentralized GPU infrastructure. This creates a unique Decentralized GPU Network for Visual Computing and AI Inference.**

***Index Terms*—Proof-of-Rendering, Active Noise Generation, Active Noise Verification, 3D/VR Rendering, Federated Learning, Inference, Burn-Mint-Work, IBME, Decentralization, GPU Compute**

I. Introduction

Inferix is a DePIN network of GPUs for visual computing and AI, it is built to bridge the needs of users and hardware owners. Its solution meets real-world problems across a range of industries, not only for the AI field but also for high-quality graphics rendering. Users (e.g. 3D graphics artists, game developers, enterprises) who need GPU computing power for rendering high-quality graphics can use Inferix system to continuously access these precious resources with faster processing time and more efficient spending. Owners of GPUs can share idle resources to Inferix network and earn long-term passive income while simultaneously balancing their main jobs or leisure activities.

At high-level, Inferix network is naturally a dynamic system where demands of digital content creators and supplies of GPU owners are created continuously over time. Users are concerned with the security and privacy of the system, with the facility of accessing computing resources, as well as with the price that they have to pay for their demands.

This section first describes the high-level architecture of Inferix network. We introduce a novel approach for the Burn-and-Mint Equilibrium [1], [2] that aims to solve the problem of precisely evaluating the value of GPU nodes up on their contribution to the network.

Next, we present briefly the graphics rendering service offered by Inferix's network and the challenges that we have to solve. In section II, we give a mathematical introduction for the Active Noise Generation and Verification algorithms, and discuss how they can be employed to solve the challenges raised by Inferix's network. We reserve section III for a detailed description of the current implementation of these algorithms. In section IV, we present the integration of this implementation into the existing layers of Inferix's network. Finally, we discuss in section V several ongoing developments in improving the robustness, performance and availability of the network.

Next, we present briefly the graphics rendering service offered by Inferix's network and the challenges that we have to solve. In section II, we give a mathematical introduction for the Active Noise Generation and Verification algorithms, and discuss how they can be employed to solve the challenges raised by Inferix's network. We reserve section III for a detailed description of the current implementation of these algorithms. In section IV, we present the integration of this implementation into the existing layers of Inferix's network. Finally, we discuss in section V several ongoing developments in improving the robustness, performance and availability of the network.

A. Burn-Mint-Work Equilibrium

We introduce in this section a novel approach for the Burn-and-Mint Equilibrium [1], [2] that aims to solve the problem of precisely evaluating the value of GPU nodes up on their contribution to the network.

$$IBM(t) = \int_0^t IB(t)\,dt$$

$$IBE(t) = \frac{1}{t}\int_0^t IB(t)\,dt$$

B. Rendering network

The graphics rendering service consists in a network of decentralized machines called *nodes* which are of 3 kinds: *manager*, *worker* and *verifier*. The *managers* and *verifiers* are dedicated machines of Inferix while the *workers* are machines joined by GPU owners. The number of *workers* is normally much larger than the number of *managers* and *verifiers*. A typical rendering session contains several steps which are shown in fig. 1:

1) A user submits a graphics scene to some *manager* using the Inferix plugin for client.
2) The *manager* who receives the graphics scene builds corresponding rendering jobs, each job consists of several parameters: range of images to be rendered, image format, etc. These job will be dispatched to *workers*.
3) Receiving a rendering job, a *worker* renders the graphics scene using the parameters given by the job. When it finishes, it sends the rendered images to a shared storage.

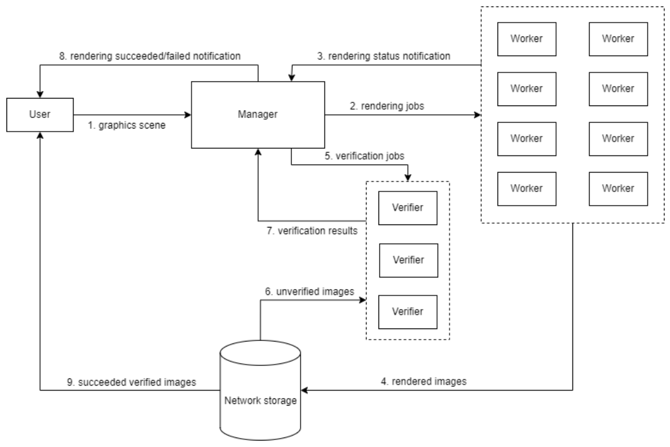

Figure 1. Graphics rendering flow

4) The *worker* then notifies a *manager* about the status of rendering job.
5) The notified manager creates a verification job and dispatches this job to some *verifier*.
6) Receiving a verification job, a *verifier* checks the authenticity of the corresponding rendered images.
7) The *verifier* then notifies the *manager* about the verification results.
8) The *manager* notifies the user about the graphics rendering result.
9) If there is no error, the user can finally download the rendered images/video from the shared storage.

The *managers* synchronize a database of rendering and verification jobs. That makes the rendering service being both logically and physically decentralized: a graphic scene can be simultaneously rendered by different *workers* and later checked by different *verifiers*, the machines of *workers* and *verifiers* can be also located at different geographical locations.

C. Rendering verification problem

A user submits some graphics work to a *manager* (cf. fig. 1), this work consists of several graphics scenes; each contains information about graphical objects, the camera, light sources and materials. The photorealistic rendering consists of sophisticated computation processes that calculate light properties at surfaces of all visible objects, results in 3D rendered images of the scene [3].

One of the most important problems that Inferix has to solve is to verify the *authenticity* of rendered results. That means how to ensure that once a user submits a valid graphics scene, then after waiting for an amount of time, the user will receive authentically rendered images. The authenticity can be defined informally as if the result received from the rendering network and the result received when the scene is genuinely rendered by a graphics rendering software are human perceptual indistinguishable.

The *workers* joined the rendering network are mostly workstations of GPU owners who want to make profit from their unused computing resources. Respecting the privacy of GPU owners and their computation resources, besides lightweight open source software installed to manage the communication with the network, there is completely no control on *workers*.

Consequently, there is no constraint to oblige *workers* to render the graphics scene correctly. Indeed, a malicious *worker* may receive a rendering job, but does nothing then uses some random images as results. Without rendering the graphics scene, the *managers* and users know only superficial features of what the rendered images look like. Obviously, the *managers* and users have no interest in rendering the scene themselves since if they can do that, there is no need to rely on *workers*. Moreover, we cannot deploy any surveillance mechanism on the machines of *worker* due to privacy reasons. Even if we try to do that, this is only a matter of time before the *worker* reverse engineers the mechanism and eventually bypasses it. The situation doesn't seem favor us: checking the authenticity of something while only having a little knowledge about it, otherwise the attacker has complete information.

Naturally, a public-key cryptography approach is using a scheme of *fully homomorphic encryption* (FHE) [4]. The graphics scene is encrypted first by a private key before sending to *workers*. Given the corresponding public key, the homomorphic encryption software performs the graphics rendering on the encrypted scene without needing to decrypt it. Finally, the encrypted rendered results are returned and decrypted at the user's side using the private key. The advantage of FHE is that the *workers*, even being able to modify the FHE softwares on their side, cannot interfere the FHE graphics rendering processes or forge the rendering results without being detected. Unfortunately, this approach is impractical since all state-of-the-art implementations will make the performance of the homomorphic encryption graphics rendering become unacceptable [5].

II. Algorithms

To handle this problem, we follow the approach of digital watermarking [6], [7] and propose a scheme called *Active Noise Generation and Verification* (ANGV) which is a variant of *proof of ownership* [8], [9]. Our scheme has several favorable properties:

1) Efficiency: noise generation and verification requires much lower computation resources compared with the graphics rendering, the total performance of the system is not affected.
2) Fidelity: the scheme needs to modify the initial scene so the rendered output will be distorted; but the distortion is regulated being below the human perception capability, hence there is no loss of quality.
3) Robustness: the embedded noises are robust under rendering enhancements (e.g. anti-aliasing) and post-processing operations.
4) Effectiveness: there is no need to use a special graphics rendering software as in the case of FHE.

A. Active noise generation and verification

In current digital watermarking schemes for authentication and ownership verification [8], [9], [10], invisible watermarks will be embedded into the digital content needed to be protected. The detector (or verifier) tries to extract the watermark from a tested content, then compares the extracted watermark with the original embedded one, if the comparison is passed then the content authenticated.

However, in the context of Inferix's rendering network, the *manager* has access to the image only after the graphics scene has been rendered by *workers*. It is nonsense to embed watermark into the image at this point since the watermarking cannot help to detect any malicious manipulations which may happen before that, i.e. in the rendering process. Our approach is to embed watermarks into the graphics scene submitted by users before sending it to *workers*. The *Active Noise Generation and Verification* consists of two algorithms as described below.

1) Noise generation: In practice, a graphics scene may contain multiple frames, each job of this scene contains some range of frames to be rendered, consequently each worker may render only a subset of these frames. For the simplification purpose, we assume in this section that a scene has only one frame, so the output image is determined uniquely by the scene. Let $\mathcal{R}$ denote the graphics rendering process, for each

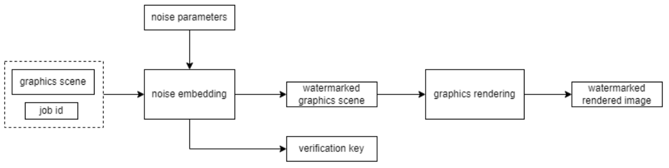

Figure 2. Noise generation

input graphics scene $\mathcal{S}$, the result of the rendering is an image:

$$I = \mathcal{R}(S) \tag{1}$$

But it is important to remark that I is actually never computed, neither by the *manager* in the watermark insertion procedure nor by *workers* in rendering processes, the equation above represents only an equality.

Similar with conventional invisible watermark schemes [8], [9], [10], a noise W is a sequence of atomic watermarks:

$$W = \{w_1, \dots, w_n\} \tag{2}$$

where w_i $(1 \leq i \leq n)$ is independently chosen from some normal probability distribution $\mathcal{N}(0, \mu^2)$. Furthermore, w_i has a special structure depending on where it is introduced in the scene S. The number n of atomic watermark signals is chosen around an experimental human perception threshold. Together with a uniformly generated job identification number $J_{\texttt{id}}$, we calculate a verification key:

$$K_{\texttt{verif}}(S, W, J_{\texttt{id}}) = \{k_1, \dots, k_n\} \tag{3}$$

that will be used later for the noise verification procedure.

The noise W is not embedded into the image I (we have mentioned above that embedding watermarks into I cannot help the authentication) but into the scene S (c.f. fig. 2). Let $\mathcal{E}$ denote the embedding function, we now create a watermarked scene:

$$\hat{S} = \mathcal{E}(S, W) \tag{4}$$

Finally $\hat{S}$ is sent to *workers* for rendering, that results in a rendered image:

$$\hat{I} = \mathcal{R}(\hat{S}) \tag{5}$$

In case of being accepted, that means $\hat{I}$ passes the noise verification which will be presented hereafter, this is the image sent back to the original user (i.e. the owner of the graphics scene $\mathcal{S}$), but not the image I in eq. (1). The encoding function $\mathcal{E}$ and the watermark W are designed so that the distortion of $\hat{I}$ against I is under human perception capability, then $\hat{I}$ can be authentically used as a result of the graphics rendering.

2) Noise verification: Different from proof of ownership schemes [8], [9], the verification of watermark requires a key. Given an image J and a verification key $K_{\texttt{repr}}$ (c.f. eq. (3)),

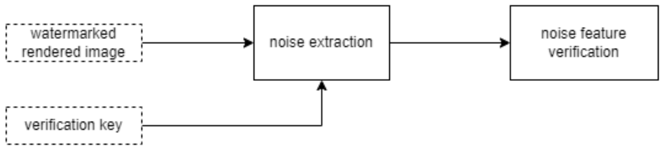

Figure 3. Noise verification

we first try to recover a watermark $\hat{W}$ from J using a decoding function $\mathcal{D}$:

$$\hat{W} = \mathcal{D}(J, K_{\texttt{repr}}) \tag{6}$$

Next $\hat{W}$ is compared against W, if the difference is above some threshold T:

$$\|\hat{W} - W\| \geq T \tag{7}$$

then J will be accepted otherwise rejected.

III. Implementation

Section II describes the high-level view of the Active Noise Generation and Verification algorithm. In this section, we present in detail the current implementation of watermark insertion and verification.

A. Structure of watermark

IV. Integration

V. Ongoing developments

References

[1] P. Snow, B. Deery, J. Lu, D. Johnston, and P. Kirby, "Factom: Business processes secured by immutable audit trails on the blockchain," Factom Protocol, Tech. Rep., 2018.

[2] A. Haleem, A. Allen, A. Thompson, M. Nijdam, and R. Garg, "Helium: A decentralized wireless network," Helium Systems, Tech. Rep., 2018.

[3] J. F. Hughes, A. van Dam, M. McGuire, D. F. Sklar, J. D. Foley, S. K. Feiner, and K. Akeley, *Computer Graphics: Principles and Practice*, 3rd ed. Addison-Wesley, 2014.

[4] C. Gentry, "Fully homomorphic encryption using ideal lattices," *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, May 2009.

[5] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. P. Fitzek, and N. Aaraj, "Survey on fully homomorphic encryption, theory, and applications," *Proceedings of the IEEE*, vol. 110, no. 10, 2022.

[6] I. J. Cox, J. Kilian, T. Leighton, and T. Shamoon, "Secure spread spectrum watermarking for multimedia," *IEEE Transactions on Image Processing*, Dec. 1997.

[7] I. J. Cox, M. L. Miller, and A. L. McKellips, "Watermarking as communications with side information," *Proceedings of the IEEE*, vol. 87, no. 7, Jul. 1999.

[8] M. Wu and B. Liu, "Watermarking for image authentication," in *Proceedings of International Conference on Image Processing*, 1998.

[9] M. M. Yeung and F. Mintzer, "An invisible watermarking technique for image verification," in *Proceedings of International Conference on Image Processing*, 1997.

[10] S. A. Craver, N. D. Memon, B.-L. Yeo, and M. M. Yeung, "Can invisible watermarks resolve rightful ownerships?" in *Storage and Retrieval for Image and Video Databases*, I. K. Sethi and R. C. Jain, Eds. SPIE, Jan. 1997.